Sommatore a 3 input (v1)

Stefano Scarcelli & Michele De Fusco

01 Dic 2023

Contents

1 Analisi progettuale **3**
1.1 Analisi preliminare 3
1.2 Struttura del progetto 3

2 Implementazione **3**
2.1 Adder n-bit 3
2.1.1 Codice VHDL 4
2.1.2 Sintesi 5
2.2 Register n-bit 5
2.2.1 Codice VHDL 5
2.2.2 Sintesi 6
2.3 Synched adder 7
2.3.1 Codice VHDL 7
2.3.2 Sintesi 8
2.4 Three adder (main) 9
2.4.1 Codice VHDL 9
2.4.2 Sintesi 11
2.5 Time constrain 11
2.5.1 Analisi del Time constrain 11

3 Testing **11**
3.1 Codice test test-bench 12
3.2 Risultati 14

1 Analisi progettuale

1.1 Analisi preliminare

L'obbiettivo è quello di costruire un circuito in grado di sommare 3 numeri a *n-bit* (*2's complements*) e restituirne il risultato. Sia gli input che gli output devono essere sincronizzati tramite l'uso di registri.
L'idea di base è quella di usare due **Ripple carry** *parametrici* in cascata tra di loro per eseguire il calcolo desiderato $A + B + C = R$.

1.2 Struttura del progetto

L'idea alla base dell'implementazione è quella di inserire in pipeline i due moduli **sommatori** per eseguire prima la somma $A + B$ e successivamente $(A + B) + C$.
Questa implementazione porta all'inserimento di (in totale) di 6 **registri**, 2 in ingresso ad ogni adder, uno in aggiunta all'input C per salvare il risultato per la seconda operazione di somma e uno in uscita per salvare il risultato dell'operazione complessiva (richiesto dalle specifiche del progetto).
Il primo ritardo per riceve un risultato coerente è di 3 colpi di clock mentre il delay per ricevere i risultati successivi al primo è di solo 1 colpo di clock.

2 Implementazione

L'implementazione si basa sulla definizione di una struttura gerarchica di componenti, partendo dalla definizione comportamentale dei componenti elementari (**Adder n-bit** e **Register n-bit**) per poi andare a comporre (tramite 2 livelli di astrazione, **Synched adder** e in fine l'elemento principale **Three adder (main)**) la struttura del progetto.

2.1 Adder n-bit

L'implementazione dell'**Adder n-bit** segue la descrizione comportamentale tramite **Ripple carry** parametrico usando i segnali *propagate* (P) e *generate* (G).

2.1.1 Codice VHDL

```
----------------------------------------------------------------------------------
-- Company: UNICAL
-- Engineer: Michele De Fusco
--
-- Create Date: 04.12.2023 14:49:19
-- Design Name: ---
-- Module Name: Adder - Version1
-- Project Name: ThreeNumbersAdder
-- Target Devices: xc7z020clg400-2
-- Tool Versions: 2023.2
-- Description: Parametric n-bit ripple carry adder
--
-- Dependencies: None
--
-- Revision: 2
-- Revision 1.0 - Implementation
-- Additional Comments: ---
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Adder is
    generic (n : integer := 8);
    Port (A, B : in STD_LOGIC_VECTOR (n-1 downto 0);
          R : out STD_LOGIC_VECTOR (n downto 0));
end Adder;

architecture Version1 of Adder is
    signal p, g : STD_LOGIC_VECTOR (n downto 0);
    signal carry : STD_LOGIC_VECTOR (n+1 downto 0);
begin
    p <= (A(n-1) xor B(n-1)) & (A xor B);
    g <= (A(n-1) and B(n-1)) & (A and B);

    carry(0) <= '0';
    carry(n+1 downto 1) <= g or (p and carry(n downto 0));
    R <= p xor carry(n downto 0);

end Version1;
```

2.1.2 Sintesi

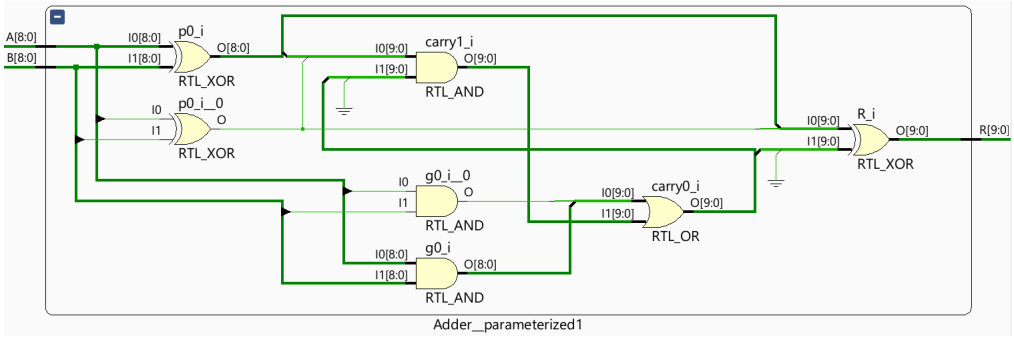

Figure 1: Circuito di **Adder**.

2.2 Register n-bit

L'implementazione dell'**Register n-bit** segue la descrizione comportamentale classica con memorizzazione a *fronti di salita*.
Il **registro** implementa in più un segnale di *clear* (asincrono) attivo alto.

2.2.1 Codice VHDL

```
----------------------------------------------------------------------------------
-- Company: UNICAL
-- Engineer: Stefano Scarcelli
--
-- Create Date: 04.12.2023 14:57:11
-- Design Name: ---
-- Module Name: Register_n - Version1
-- Project Name: ThreeNumbersAdder
-- Target Devices: xc7z020clg400-2
-- Tool Versions: 2023.2
-- Description: Parametric n-bit register
--
-- Dependencies: None
--
-- Revision: 1
-- Revision 1.0 - Implementation
-- Additional Comments: ---
--
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Register_n is
    generic (n : integer := 8);
    Port (CLK, Clear : in STD_LOGIC;
          D : in STD_LOGIC_VECTOR (n-1 downto 0);
          Q : out STD_LOGIC_VECTOR (n-1 downto 0));
end Register_n;

architecture Version1 of Register_n is

begin
    process(CLK, Clear) begin
        if (Clear = '1') then
            Q <= (others => '0');
        elsif rising_edge(CLK) then
            Q <= D;
        end if;
    end process;

end Version1;
```

2.2.2 Sintesi

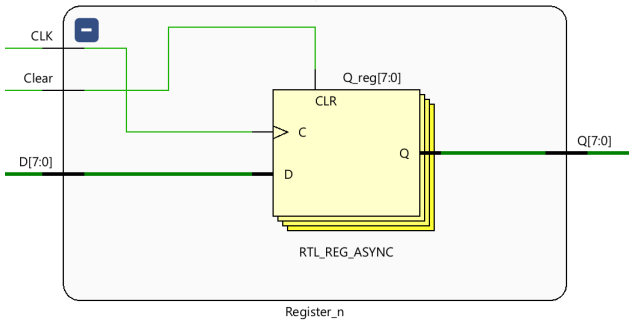

Figure 2: **Registro** a 8bit.

2.3 Synched adder

Questo è un componente intermedio che abbiamo impostato per poi costruire il circuito completo.
Consiste in un **Adder n-bit** con 2 **registri** collocati agli input di esso.

2.3.1 Codice VHDL

```
----------------------------------------------------------------------------------
-- Company: UNICAL
-- Engineer: Stefano Scarcelli, Michele De Fusco
--
-- Create Date: 04.12.2023 15:31:55
-- Design Name: ---
-- Module Name: Synched_adder - Version1
-- Project Name: ThreeNumbersAdder
-- Target Devices: xc7z020clg400-2
-- Tool Versions: 2023.2
-- Description: Adder with input registers
--
-- Dependencies: Adder.vhd, Register_n.vhd
--
-- Revision: 1
-- Revision 1.0 - Implementation
-- Additional Comments: ---
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Synched_adder is
    generic (n : integer := 8);
    Port (A, B : in STD_LOGIC_VECTOR (n-1 downto 0);
          R : out STD_LOGIC_VECTOR (n downto 0);
          CLK, Clear : in STD_LOGIC);
end Synched_adder;

architecture Version1 of Synched_adder is
    component Adder is
        generic (n : integer := 8);
        Port (A, B : in STD_LOGIC_VECTOR (n-1 downto 0);
              R : out STD_LOGIC_VECTOR (n downto 0));
```

```
    end component;
    component Register_n is
        generic (n : integer := 8);
        Port (CLK, Clear : in STD_LOGIC;
              D : in STD_LOGIC_VECTOR (n-1 downto 0);
              Q : out STD_LOGIC_VECTOR (n-1 downto 0));
    end component;
    signal Ra, Rb: STD_LOGIC_VECTOR (n-1 downto 0);
begin
    RegA: Register_n generic map(n) port map(CLK, Clear, A, Ra);
    RegB: Register_n generic map(n) port map(CLK, Clear, B, Rb);

    Adder1: Adder generic map(n) port map(Ra, Rb, R);

end Version1;
```

2.3.2 Sintesi

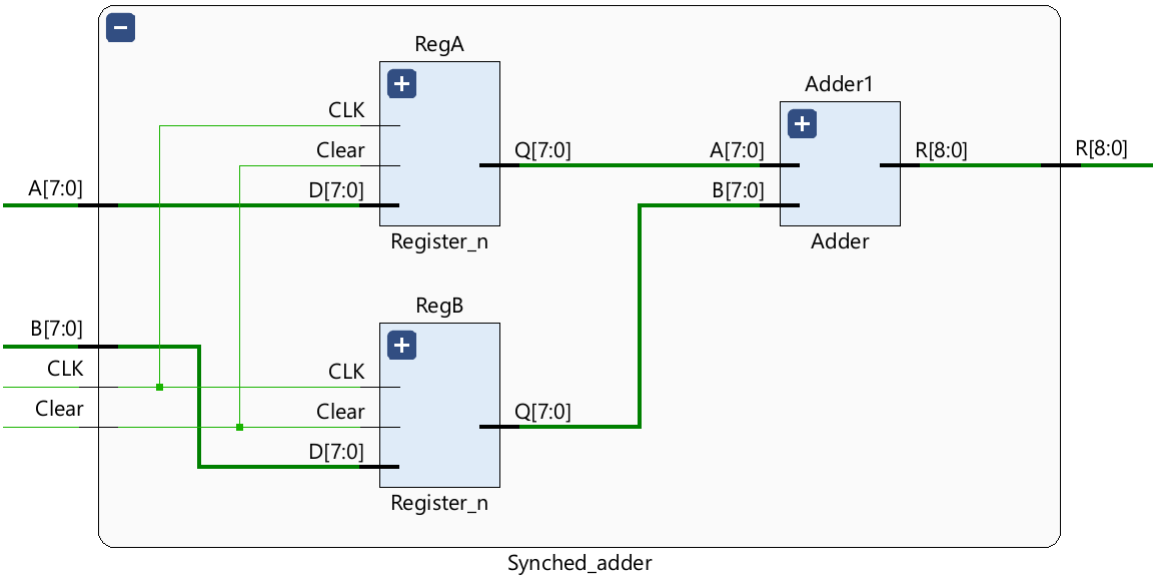

Figure 3: Circuito del **Synched adder**.

2.4 Three adder (main)

Il circuito finale comprende invece l'uso di 2 **Synched adder** e 2 **registri** aggiuntivi, uno collocato tra l'input C del circuito e il secondo ingresso del secondo **Synched adder** (usato come buffer per salvare il valore di C nella pipeline) e un'altro tra il risultato dell'operazione e l'output del circuito (come richiesto dalle specifiche del progetto).

2.4.1 Codice VHDL

```
----------------------------------------------------------------------------------
-- Company: UNICAL
-- Engineer: Stefano Scarcelli, Michele De Fusco
--
-- Create Date: 04.12.2023 15:46:04
-- Design Name: ---
-- Module Name: Three_Adder - Version1
-- Project Name: ThreeNumbersAdder
-- Target Devices: xc7z020clg400-2
-- Tool Versions: 2023.2
-- Description: Main file of the project
--
-- Dependencies: Register_n.vhd, Synched_adder.vhd
--
-- Revision: 1
-- Revision 1.0 - Implementation
-- Additional Comments: ---
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Three_Adder is
    generic (n : integer := 8);
    Port (A : in STD_LOGIC_VECTOR (n-1 downto 0);
          B : in STD_LOGIC_VECTOR (n-1 downto 0);
          C : in STD_LOGIC_VECTOR (n-1 downto 0);
          R : out STD_LOGIC_VECTOR (n+1 downto 0);
          CLK : in STD_LOGIC;
          Clear : in STD_LOGIC);
end Three_Adder;
```

```
architecture Version1 of Three_Adder is
    component Synched_adder
        generic (n : integer := 8);
        Port (A, B : in STD_LOGIC_VECTOR (n-1 downto 0);
              R : out STD_LOGIC_VECTOR (n downto 0);
              CLK, Clear : in STD_LOGIC);
    end component;
    component Register_n
        generic (n : integer := 8);
        Port (CLK, Clear : in STD_LOGIC;
              D : in STD_LOGIC_VECTOR (n-1 downto 0);
              Q : out STD_LOGIC_VECTOR (n-1 downto 0));
    end component;
    component Adder
        generic (n : integer := 8);
        Port (A, B : in STD_LOGIC_VECTOR (n-1 downto 0);
              R : out STD_LOGIC_VECTOR (n downto 0));
    end component;
    signal Rc: STD_LOGIC_VECTOR (n-1 downto 0);
    signal Rs,RCext: STD_LOGIC_VECTOR (n downto 0);
    signal RSF: STD_LOGIC_VECTOR (n+1 downto 0);
begin
    RCext <= Rc(n-1) & Rc;
    RegC: Register_n generic map(n) port map(CLK, Clear, C, Rc);

    SA1:  Synched_adder generic map(n) port map(A, B, Rs, CLK, Clear);
    SA2:  Synched_adder generic map(n+1) port map(Rs, RCext, RSF, CLK, Clear);

    RegSF:Register_n generic map(n+2) port map(CLK, Clear, RSF, R);

end Version1;
```

2.4.2 Sintesi

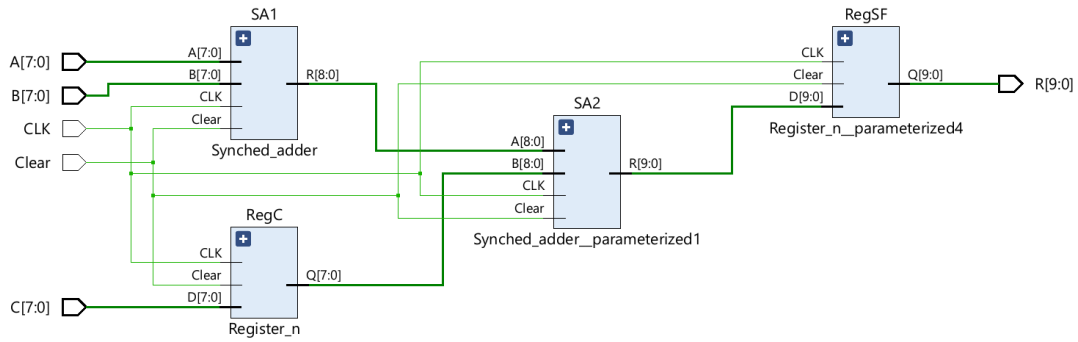

Figure 4: Circuito principale.

2.5 Time constrain

In più abbiamo aggiunto un *time constrain* sul segnale di clock con un periodo di **5ns**.

```
create_clock -period 5.000 -name main_clock
-waveform {0.000 2.500} [get_nets CLK]
```

2.5.1 Analisi del Time constrain

Con l'implementazione del codice abbiamo verificato (*Timing summary routed*) che il **Time constrain** fosse valido, valutando di aver inserito un periodo di clock di superiore rispetto al minimo tollerabile dal circuito di **1.958ns**.

3 Testing

Per il test abbiamo optato ad una analisi parziale eseguendo somme di numeri che potessero stressare al massimo i componenti circuitali per verificarne il corretto funzionamento.

3.1 Codice test test-bench

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: Stefano Scarcelli, Michele De Fusco
--
-- Create Date: 12.12.2023 11:20:24
-- Design Name: ---
-- Module Name: Sim_Add - Version1
-- Project Name: ThreeNumbersAdder
-- Target Devices: xc7z020clg400-2
-- Tool Versions: 2023.2
-- Description: Main simulation
--
-- Dependencies: Three_Adder.vhd
--
-- Revision: 2
-- Revision 1.0 - Implementation
-- Additional Comments: ---
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.ALL;

entity Sim_Add is
    generic (n : integer := 8);
end Sim_Add;

architecture Version1 of Sim_Add is
    component Three_Adder
        --generic (n : integer := 8);
        Port (A : in STD_LOGIC_VECTOR (n-1 downto 0);
              B : in STD_LOGIC_VECTOR (n-1 downto 0);
              C : in STD_LOGIC_VECTOR (n-1 downto 0);
              R : out STD_LOGIC_VECTOR (n+1 downto 0);
              CLK : in STD_LOGIC;
              Clear : in STD_LOGIC);
    end component;
    Signal IA, IB, IC : STD_LOGIC_VECTOR (n-1 downto 0);
    Signal ORR : STD_LOGIC_VECTOR (n+1 downto 0);
    signal clk, clear : STD_LOGIC;
    constant T : time := 10 ns;
```

```
begin
    TA : Three_Adder port map(IA, IB, IC, ORR, clk, clear);
    process begin
        clk <= '0';
        wait for T/2;
        clk <= '1';
        wait for T/2;
    end process;

    process begin
        wait for 100 ns;

        clear <= '1';
        wait for T;
        clear <= '0';
        wait for T/4;

        -- (-1)+(-1)+(-1)=-3
        IA <= (others=>'1');
        IB <= (others=>'1');
        IC <= (others=>'1');
        wait for T;

        -- 127+127+127=381
        IA(n-1) <= ('0');
        IA(n-2 downto 0) <= (others=>'1');
        IB(n-1) <= ('0');
        IB(n-2 downto 0) <= (others=>'1');
        IC(n-1) <= ('0');
        IC(n-2 downto 0) <= (others=>'1');
        wait for T;

        -- (-64)+1+(-63)=0
        IA(n-1 downto n-2) <= (others=>'0');
        IA(n-3 downto 0) <= (others=>'1');
        IB(n-1 downto 1) <= (others=>'0');
        IB(0) <= ('1');
        IC(n-1 downto n-2) <= (others=>'1');
        IC(n-3 downto 0) <= (others=>'0');
        wait for T;

        -- Waiting results
        wait for T*2;
    end process;
```

```
end Version1;
```

3.2 Risultati

I risultati della simulazione post implementativa confermano il funzionamento del circuito come previsto.

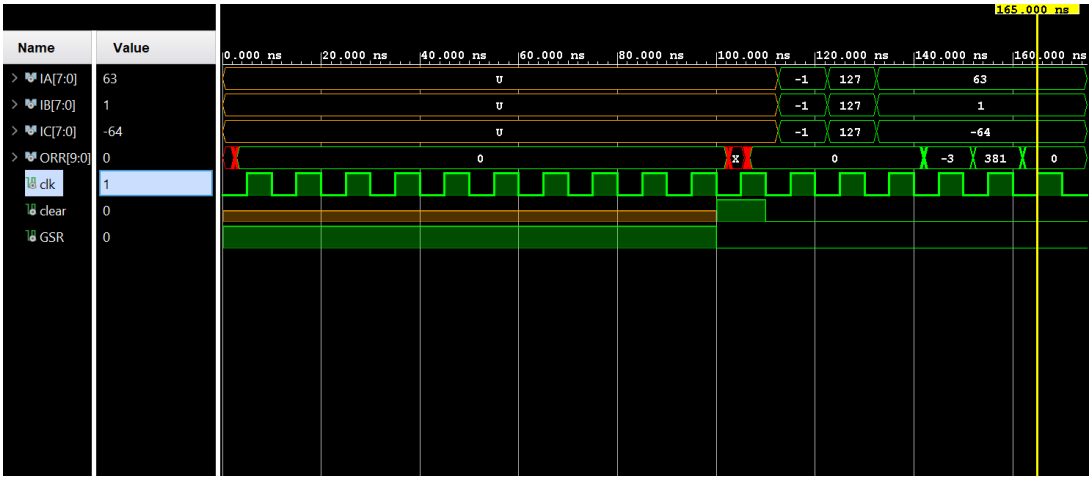

Figure 5: Grafico temporale simulazione post implementazione.